



“Lanza y Aprende”: Actividad guiada de Movimiento Parabólico (HTML5 + Bootstrap 5 + CSS + JS)

Modalidad: trabajo en equipo (3 personas)

Propósito: comprender el tiro parabólico construyendo, ejecutando y explicando una simulación web con HTML5, Bootstrap 5, CSS y JavaScript.

Organización del equipo (3 roles)

- **Frontender:** estructura HTML y Bootstrap.
- **Styler:** aplica la hoja de estilos CSS.
- **Coder físico:** conecta y explica el JavaScript (fórmulas y animación).

Objetivos de aprendizaje

1. Identificar elementos HTML5 y clases de Bootstrap 5.
2. Aplicar estilos básicos en una hoja CSS externa.
3. Relacionar variables físicas del tiro (v_0 , ángulo, g) con una **trayectoria** en JS.
4. Ejecutar y explicar una **simulación** de movimiento parabólico.

✓ Entregable final

- Carpeta del proyecto con **index.html**, **styles.css** y **app3.js**.
- Respuestas a preguntas que estarán localizadas en el documento
- Un **informe breve** (2 páginas) que explique:
 - qué modifica cada archivo,
 - cómo cambian los resultados al variar **v₀**, **ángulo** y **g**,
 - una captura o GIF corto de la simulación corriendo.

📁 Paso 1. Estructura de carpetas

Crea una carpeta, por ejemplo `mov-parabolico/`, con **tres archivos** al mismo nivel:

```
mov-parabolico/  
├─ index.html  
├─ styles.css  
└─ app3.js
```

🧱 Paso 2. HTML base con Bootstrap (archivo `index.html`)

Copia **exactamente** el siguiente contenido en `index.html`:

```
<!DOCTYPE html>  
<html lang="es">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-  
scale=1.0">  
  <title>Movimiento Parabólico</title>  
  
  <!-- Bootstrap 5 -->  
  <link  
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstr  
ap.min.css" rel="stylesheet">  
  <!-- Estilos propios -->  
  <link rel="stylesheet" href="styles.css">  
</head>  
<body class="bg-light">
```

```

<div class="container text-center mt-4">
  <h1 class="text-primary">🚀 Simulación de Movimiento
Parabólico</h1>
  <p class="text-muted">Proyecto grupal con HTML5, CSS, JS y
Bootstrap 5</p>

  <!-- Contenedor para el Canvas -->
  <div class="card shadow p-3">
    <canvas id="plano" width="800" height="400"></canvas>
  </div>

  <!-- Botón de simulación -->
  <div class="mt-3">
    <!--
    <button class="btn btn-primary"
onclick="animarProyectil()">🚀 Iniciar Simulación</button>
    -->
    <button class="btn btn-primary"
onclick="iniciarSimulacion()">🚀 Iniciar Simulación</button>
  </div>
</div>

<!-- Panel de parámetros -->
<div class="container mt-4">
  <div class="row">
    <!-- Parámetros -->
    <div class="col-md-4">
      <div class="card p-3 shadow-sm">
        <h5 class="text-primary">⚙️ Parámetros</h5>
        <label>Velocidad inicial (v0)</label>
        <input type="number" id="inputV0" class="form-
control" value="60">

        <label class="mt-2">Ángulo (°)</label>
        <input type="number" id="inputAngle"
class="form-control" value="45">

        <label class="mt-2">Gravedad (g)</label>
        <input type="number" id="inputG" class="form-
control" value="9.8">

```

```

        <button class="btn btn-success mt-3"
onclick="actualizarParametros()">Actualizar</button>
    </div>
</div>

<!-- Datos de salida -->
<div class="col-md-8">
    <div class="card p-3 shadow-sm">
        <h5 class="text-success">📊 Resultados</h5>
        <p id="datos">Modifica parámetros y lanza el
proyecil 🚀 </p>
    </div>
</div>
</div>
</div>

<!-- Script Bootstrap -->
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap
.bundle.min.js"></script>
<!-- Librería GSAP -->
<script
src="https://cdn.jsdelivr.net/npm/gsap@3.12.2/dist/gsap.min.js">
</script>
<!-- Script propio -->
<script src="app3.js"></script>
</body>
</html>

```

Qué hace este HTML (explicación breve para ustedes):

- Importa **Bootstrap 5** y tu **styles.css**.
- Crea un **canvas** (`id="plano"`) donde se dibuja el plano y la trayectoria.
- Muestra un botón “🚀 Iniciar Simulación” que llama a `iniciarSimulacion()`.
- Agrega un panel para **parámetros** (v_0 , ángulo, g) y un panel para **resultados**.
- Carga al final **app3.js** (lógica y animación).

Paso 3. Estilos CSS (archivo `styles.css`)

Copia **exactamente** este contenido en `styles.css`:

```
/* Fondo del canvas */
#plano {
  border: 2px solid #0d6efd; /* Azul Bootstrap */
  background-color: #f8f9fa; /* Gris claro */
}

/* Estilo general */
body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

/* Card donde está el canvas */
.card {
  max-width: 820px;
  margin: auto;
}
```

Qué logra el CSS:

- Delimita y resalta el **canvas** con borde azul y fondo claro.
- Define una fuente legible para todo el sitio.
- Centra y limita el ancho de la **card** del canvas.

Paso 4. Lógica y animación (archivo `app3.js`)

Copia **exactamente** este contenido en `app3.js`:

```
// =====
// Configuración inicial del canvas
// =====
const canvas = document.getElementById("plano");
const ctx = canvas.getContext("2d");

// =====
// Dibujar plano cartesiano
// =====
```

```

function dibujarPlano() {
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    ctx.strokeStyle = "#000";
    ctx.lineWidth = 2;

    // Eje x
    ctx.beginPath();
    ctx.moveTo(40, canvas.height - 40);
    ctx.lineTo(canvas.width - 20, canvas.height - 40);
    ctx.stroke();

    // Eje y
    ctx.beginPath();
    ctx.moveTo(40, canvas.height - 20);
    ctx.lineTo(40, 20);
    ctx.stroke();

    // Flechas
    ctx.beginPath();
    ctx.moveTo(canvas.width - 25, canvas.height - 45);
    ctx.lineTo(canvas.width - 20, canvas.height - 40);
    ctx.lineTo(canvas.width - 25, canvas.height - 35);
    ctx.fill();

    ctx.beginPath();
    ctx.moveTo(35, 25);
    ctx.lineTo(40, 20);
    ctx.lineTo(45, 25);
    ctx.fill();

    // Etiquetas
    ctx.font = "14px Arial";
    ctx.fillText("X", canvas.width - 30, canvas.height - 50);
    ctx.fillText("Y", 25, 30);
}

// Dibujar plano al cargar
dibujarPlano();

// =====
// variables físicas

```

```
// =====
let v0 = 60;      // velocidad inicial
let angle = 45;   // Ángulo en grados
let g = 9.8;      // Gravedad

let rad = toRadians(angle);
let vx = v0 * Math.cos(rad);
let vy = v0 * Math.sin(rad);

// variables para animación
let animando = false;
let startTime;
let trayectoria = []; // Array para guardar puntos de la
trayectoria

// =====
// Funciones auxiliares
// =====
function toRadians(angle) {
    return angle * (Math.PI / 180);
}

function position(t) {
    let x = vx * t;
    let y = vy * t - 0.5 * g * t * t;
    return {x, y};
}

// =====
// Dibujar proyectil y rastro
// =====
function dibujarEscena(x, y) {
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    dibujarPlano();

    // Dibujar trayectoria completa
    ctx.beginPath();
    ctx.strokeStyle = "rgba(0, 123, 255, 0.7)"; // Azul
    semitransparente
    ctx.lineWidth = 2;
    for (let i = 0; i < trayectoria.length; i++) {
        let punto = trayectoria[i];
```

```

        if (i === 0) {
            ctx.moveTo(40 + punto.x, canvas.height - 40 - punto.y);
        } else {
            ctx.lineTo(40 + punto.x, canvas.height - 40 - punto.y);
        }
    }
    ctx.stroke();

    // Dibujar proyectil
    ctx.beginPath();
    ctx.arc(40 + x, canvas.height - 40 - y, 8, 0, 2 * Math.PI);
    ctx.fillStyle = "orange"; // color llamativo
    ctx.shadowColor = "rgba(255, 165, 0, 0.7)";
    ctx.shadowBlur = 15; // efecto brillante
    ctx.fill();
}

// =====
// Animación con requestAnimationFrame
// =====
function animarProyectil(timestamp) {
    if (!startTime) startTime = timestamp;
    let elapsed = (timestamp - startTime) / 1000; // segundos

    let pos = position(elapsed);

    if (pos.y >= 0) {
        trayectoria.push(pos); // Guardar cada punto en el array
        dibujarEscena(pos.x, pos.y);
        requestAnimationFrame(animarProyectil);
    } else {
        animando = false;
    }
}

// =====
// Lanzar simulación
// =====
function iniciarSimulacion() {
    if (!animando) {
        startTime = null;
        animando = true;
    }
}

```



```

        trayectoria = []; // Reiniciar trayectoria
        requestAnimationFrame(animarProyectil);
    }
}

// =====
// Actualizar parámetros
// =====
function actualizarParametros() {
    v0 = parseFloat(document.getElementById("inputV0").value);
    angle =
parseFloat(document.getElementById("inputAngle").value);
    g = parseFloat(document.getElementById("inputG").value);

    rad = toRadians(angle);
    vx = v0 * Math.cos(rad);
    vy = v0 * Math.sin(rad);

    document.getElementById("datos").innerHTML = `
        <b>Velocidad:</b> ${v0} <br>
        <b>Ángulo:</b> ${angle}° <br>
        <b>vx:</b> ${vx.toFixed(2)} px/s <br>
        <b>vy:</b> ${vy.toFixed(2)} px/s <br>
        <b>Gravedad:</b> ${g} m/s²
    `;
}

```

Qué hace el JS (en lenguaje claro):

- Dibuja un **plano cartesiano** en el canvas y lo **refresca** en cada frame.
- Define variables físicas: v_0 , **ángulo** y g . Convierte el ángulo a radianes, y calcula v_x y v_y .
- Usa la fórmula $y(t) = v_y \cdot t - \frac{1}{2} \cdot g \cdot t^2$, $x(t) = v_x \cdot t$ para la trayectoria.
- Registra el **rastro** de puntos en `trayectoria` y lo dibuja como línea.
- Controla la animación con `requestAnimationFrame`.
- Permite **actualizar parámetros** desde los inputs y ver los valores calculados.


Paso 5. Relación ciencia–código (mini guía)

- v_0 (velocidad inicial): define qué tan “rápido” sale el proyectil.
- **Ángulo** ($^\circ$): entre 0° y 90° . Valores cercanos a 45° maximizan el alcance (en condiciones ideales).
- **g** (gravedad): en la Tierra, $\sim 9,8 \text{ m/s}^2$; valores mayores “aplastan” la trayectoria.

Paso 6. Conexión entre archivos

- `index.html` **incluye** `styles.css` (estilos) y **carga** `app3.js` al final para que el DOM esté listo. index
- `app3.js` **busca** el canvas `#plano` que definiste en el HTML para dibujar. app3
- `styles.css` **decora** el canvas y la página. styles

Paso 7. Prueba local

- Abre `index.html` con tu navegador (doble clic).
- Pulsa “ **Iniciar Simulación**” y observa el arco.
- Cambia v_0 , **ángulo** y **g**, presiona “**Actualizar**” y vuelve a lanzar.

Paso 8. Micro–experimentos en equipo

1. **Ángulo fijo** (45°), varía v_0 (30, 60, 90). ¿Cómo cambia el **alcance**?
2. v_0 **fijo** (60), varía **ángulo** (20° , 45° , 70°). ¿Cuál llega más lejos?
3. v_0 y **ángulo fijos**, varía **g** (Luna ~ 1.6 , Tierra 9.8, Júpiter ~ 24.8). ¿Cómo cambia la curva?

(Usen el panel de parámetros y describan lo observado en el informe.)

Paso 9. Lee el código y explica (en tu informe)

- ¿Qué hace `toRadians` y por qué se necesita? app3
- ¿Cómo se construye el **rastro** de la trayectoria (`trayectoria.push(pos)`)? app3
- ¿Por qué la animación se detiene cuando `pos.y < 0`? app3

Paso 10. Interfaz clara para el usuario

- Identifica visualmente dónde están **parámetros** y **resultados** (cards Bootstrap). index
- Explica para qué sirve el botón “**Actualizar**” y cuándo presionarlo. app3

Paso 11. Roles y registro de cambios (bitácora)

- Cada equipo anota **qué cambió** (v_0 , ángulo, g) y **qué pasó** (más/menos alcance, mayor altura, etc.).
- Incluye 3 capturas (o un GIF) de **tres escenarios distintos**.

Paso 12. Reflexión final (¡pensamiento crítico!)

- ¿Qué suposiciones tiene este modelo (sin rozamiento, sin viento)?
- ¿Cómo afectaría el resultado si incluyeras **altura inicial** o **resistencia del aire**?

Paso 13 (opcional): Ampliaciones guiadas (sin cambiar el código base)

En el informe, *imaginen* cómo agregarían una marca del **punto máximo** (vértice) o el **alcance** sin modificar estos archivos. (Solo explicación conceptual).

Rúbrica corta (10 puntos)

- **Ejecución técnica (4)**: proyecto corre, se ve plano y trayectoria, usa parámetros.
- **Informe (4)**: explica qué hace cada archivo, respuestas a los micro-experimentos, reflexión.
- **Trabajo en equipo (2)**: roles claros, bitácora de cambios.

